RISE

RISC-V Software Ecosystem

Optimizing Software for RISC-V

Video Dev Days Nov 3, 2024

Nathan Egge negge@google.com

people.videolan.org/~negge/vdd24.pdf

Who am I?

- Engineering Manager in Android
    - TL of Native Tools and Libraries team
    - Compilers, toolchains, external libraries, NDK, etc.
- Co-author of AV1 format, worked on Daala and Theora before that
- Member of multimedia OSS non-profits: Xiph.Org and VideoLAN Asso
- Co-chair of the Technical Steering Committee in RISE non-profit

Who am I?

- Engineering Manager in Android
  - TL of Native Tools and Libraries team
  - Compilers, toolchains, external libraries, NDK, etc.
- Co-author of AV1 format, worked on Daala and Theora before that
- Member of multimedia OSS non-profits: Xiph.Org and VideoLAN Asso
- Co-chair of the Technical Steering Committee in RISE non-profit
- RISE = RISC-V Software Ecosystem [1]
  - Mission: *Accelerate the development of open source software for RISC-V*

[1] https://riseproject.dev

RISE Case Study: Adding RVV 1.0 to dav1d

- dav1d is an AV1 decoder
  - Goals: fastest software decoder, cross-platform, small binary size
  - Achieves this through a *LOT* of handwritten assembly!

```
Totals grouped by language (dominant language first):
asm:          241720 (85.05%)
ansic:         42309 (14.89%)
sh:              172 (0.06%)
```

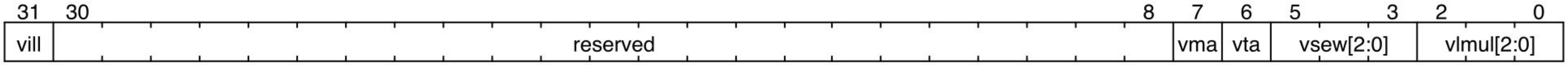
  - Essentially a C orchestrator around specialized DSP functions
    - C ABI conformance matters, assembly functions can do weird things
  - Good place to test RISC-V Vector assembly routines
    - Excellent testing framework with built-in performance evaluation

[1] https://code.videolan.org/videolan/dav1d/

RISC-V Vector (RVV) SIMD at a Glance

- Scalable Vector Implementation
  - Implementation dependent, VLEN = vector length from 128 to 16384
  - In practice, application processors could have 128, 256, 512 or 1024

- SIMD lane size configurable at run time
  - SEW = Selected Element Width from 8-bit to 64-bit
  - Set using the `vsetvli`, `vsetivli` or `vsetvl` instruction, state maintained internally



Note | This diagram shows the layout for RV32 systems, whereas in general `vill` should be at bit XLEN-1.

- Operations are generally element size “agnostic”
  - Instructions element size “agnostic”
    - Just one `vadd.vv` instead of `vadd.i8`, `vadd.i16`, `vadd.i32`, `vadd.i64` variations
  - Widening operations of the form SEW <op> SEW -> 2*SEW, e.g., `vwadd.vv`
    - Three operand instructions for fused multiply-add, e.g., `vwmacc.vv`
  - Narrowing operations go 2*SEW -> SEW, e.g, `vnsra.wi` or `vnclipu.wi`

RISC-V Vector SIMD in dav1d (503 of 3566)

- itx
  - inv_txfm_add_4x4*_8bpc (34)
  - inv_txfm_add_4x8*_8bpc (32)
  - inv_txfm_add_4x16*_8bpc (48)
  - inv_txfm_add_8x4*_8bpc (32)
  - inv_txfm_add_8x8*_8bpc (32)
  - inv_txfm_add_8x16*_8bpc (48)
  - inv_txfm_add_16x4*_8bpc (48)
  - inv_txfm_add_16x8*_8bpc (48)
  - inv_txfm_add_16x16*_8bpc (36)

- pal
  - pal_idx_finish* (5)
  - pal_pred*_8bpc (5)
  - pal_pred*_16bpc (5)

- cdef
  - cdef_filter_4x4*8bpc (3)
  - cdef_filter_4x8*8bpc (3)
  - cdef_filter_8x8*8bpc (3)
  - cdef_filter_4x4*16bpc (3)
  - cdef_filter_4x8*16bpc (3)
  - cdef_filter_8x8*16bpc (3)

- cfl
  - cfl_pred_cfl*_8bpc (4)
  - cfl_pred_cfl_128*_8bpc (4)
  - cfl_pred_cfl_left*_8bpc (4)
  - cfl_pred_cfl_top*_8bpc (4)
  - cfl_pred_cfl*_8bpc (4)
  - cfl_pred_cfl_128*_16bpc (4)
  - cfl_pred_cfl_left*_16bpc (4)
  - cfl_pred_cfl_top*_16bpc (4)

RISC-V Vector SIMD in dav1d (503 of 3566)

- ipred
  - intra_pred_paeth*_8bpc (5)
  - intra_pred_smooth*_8bpc (5)
  - intra_pred_smooth_h*_8bpc (5)
  - intra_pred_smooth_v*_8bpc (5)
  - intra_pred_paeth*_16bpc (5)
  - intra_pred_smooth*_16bpc (5)
  - intra_pred_smooth_h*_16bpc (5)
  - intra_pred_smooth_v*_16bpc (5)

- mc
  - blend*_8bpc (4)
  - blend_h*_8bpc (7)
  - blend_v*_8bpc (5)
  - blend*_16bpc (4)
  - avg*_8bpc (6)
  - w_avg*8bpc (6)
  - mask*_8bpc (6)
  - warp*_8bpc (2)

RISC-V Vector SIMD in dav1d (503 of 3566)

- ipred
  - intra_pred_paeth*_8bpc (5)
  - intra_pred_smooth*_8bpc (5)
  - intra_pred_smooth_h*_8bpc (5)
  - intra_pred_smooth_v*_8bpc (5)
  - intra_pred_paeth*_16bpc (5)
  - intra_pred_smooth*_16bpc (5)
  - intra_pred_smooth_h*_16bpc (5)
  - intra_pred_smooth_v*_16bpc (5)

- mc
  - blend*_8bpc (4)
  - blend_h*_8bpc (7)
  - blend_v*_8bpc (5)
  - blend*_16bpc (4)
  - avg*_8bpc (6)
  - w_avg*8bpc (6)
  - mask*_8bpc (6)
  - warp*_8bpc (2)

Let’s take a closer look

Example: 8bpc mc blend

```
#define blend_px(a, b, m) (((a * (64 - m) + b * m) + 32) >> 6)
static void blend_c(uint8_t *dst, const ptrdiff_t dst_stride, const uint8_t *tmp,
                    const int w, int h, const uint8_t *mask) {
    for (; h-- > 0; dst += dst_stride, tmp += w, mask += w)
        for (int x = 0; x < w; x++) dst[x] = blend_px(dst[x], tmp[x], mask[x]);
}
```

Pseudo-code for RVV, note `w` can be 4, 8, 16 or 32 only

Set VL based on `w`

Load vectors for `dst`, `tmp` and `mask`

Scratch vector for widening multiply, followed by widening multiply accumulate

Narrowing shift with rounding

Store back into `dst`

Example: 8bpc mc blend (outer loop)

```
#define blend_px(a, b, m) (((a * (64 - m) + b * m) + 32) >> 6)
static void blend_c(uint8_t *dst, const ptrdiff_t dst_stride, const uint8_t *tmp,
                    const int w, int h, const uint8_t *mask) {
    for (; h—- > 0; dst += dst_stride, tmp += w, mask += w)
        ~~for (int x = 0; x < w; x++) dst[x] = blend_px(dst[x], tmp[x], mask[x]);~~
}

function blend_8bpc_rvv, export=1, ext="v”




  ret
endfunc
```

Example: 8bpc mc blend (outer loop)

```
#define blend_px(a, b, m) (((a * (64 - m) + b * m) + 32) >> 6)
static void blend_c(uint8_t *dst, const ptrdiff_t dst_stride, const uint8_t *tmp,
                    const int w, int h, const uint8_t *mask) {
    for (; h—- > 0; dst += dst_stride, tmp += w, mask += w)
        ~~for (int x = 0; x < w; x++) dst[x] = blend_px(dst[x], tmp[x], mask[x]);~~
}
```

```
function blend_8bpc_rvv, export=1, ext="v”
  vsetvli zero, a3, e8, m1, ta, ma



  ret
endfunc
```

Example: 8bpc mc blend (outer loop)

```
#define blend_px(a, b, m) (((a * (64 - m) + b * m) + 32) >> 6)
static void blend_c(uint8_t *dst, const ptrdiff_t dst_stride, const uint8_t *tmp,
                    const int w, int h, const uint8_t *mask) {
    for (; h-- > 0; dst += dst_stride, tmp += w, mask += w)
        ~~for (int x = 0; x < w; x++) dst[x] = blend_px(dst[x], tmp[x], mask[x]);~~
}

function blend_8bpc_rvv, export=1, ext="v”
  vsetvli zero, a3, e8, m1, ta, ma
  li t1, 64                // t1 = 64;








  ret
endfunc
```

Example: 8bpc mc blend (outer loop)

```
#define blend_px(a, b, m) (((a * (64 - m) + b * m) + 32) >> 6)
static void blend_c(uint8_t *dst, const ptrdiff_t dst_stride, const uint8_t *tmp,
                    const int w, int h, const uint8_t *mask) {
    for (; h-- > 0; dst += dst_stride, tmp += w, mask += w)
        ~~for (int x = 0; x < w; x++) dst[x] = blend_px(dst[x], tmp[x], mask[x]);~~
}

function blend_8bpc_rvv, export=1, ext="v"
  vsetvli zero, a3, e8, m1, ta, ma
  li t1, 64                  // t1 = 64;
1:                           // do {
  addi a4, a4, -1            //   h = h - 1;
  ...




  bnez a4, 1b                // } while (h != 0)
  ret
endfunc
```

Example: 8bpc mc blend (outer loop)

```
#define blend_px(a, b, m) (((a * (64 - m) + b * m) + 32) >> 6)
static void blend_c(uint8_t *dst, const ptrdiff_t dst_stride, const uint8_t *tmp,
                    const int w, int h, const uint8_t *mask) {
    for (; h-- > 0; dst += dst_stride, tmp += w, mask += w)
        ~~for (int x = 0; x < w; x++) dst[x] = blend_px(dst[x], tmp[x], mask[x]);~~
}

function blend_8bpc_rvv, export=1, ext="v”
  vsetvli zero, a3, e8, m1, ta, ma
  li t1, 64                     // t1 = 64;
1:                              // do {
  addi a4, a4, -1               //   h = h - 1;
  ...
  add a0, a0, a1                // dst += dst_stride
  add a2, a2, a3                // tmp += w;
  add a5, a5, a3                // mask += w;
  bnez a4, 1b                   // } while (h != 0)
  ret
endfunc
```

RISE

Example: 8bpc mc blend (inner loop)

```
#define blend_px(a, b, m) (((a * (64 - m) + b * m) + 32) >> 6)
static void blend_c(uint8_t *dst, const ptrdiff_t dst_stride, const uint8_t *tmp,
                    const int w, int h, const uint8_t *mask) {
    ~~for (; h-- > 0; dst += dst_stride, tmp += w, mask += w)~~
        for (int x = 0; x < w; x++) dst[x] = blend_px(dst[x], tmp[x], mask[x]);
}
```

Example: 8bpc mc blend (inner loop)

```
#define blend_px(a, b, m) (((a * (64 - m) + b * m) + 32) >> 6)
static void blend_c(uint8_t *dst, const ptrdiff_t dst_stride, const uint8_t *tmp,
                    const int w, int h, const uint8_t *mask) {
    ~~for (; h-- > 0; dst += dst_stride, tmp += w, mask += w)~~
        for (int x = 0; x < w; x++) dst[x] = blend_px(dst[x], tmp[x], mask[x]);
}
```

```
  vle8.v v0, (a0)           // v0 = *dst;
  vle8.v v4, (a2)           // v4 = *tmp;
  vle8.v v8, (a5)           // v8 = *mask;
```

RISE

Example: 8bpc mc blend (inner loop)

```
#define blend_px(a, b, m) (((a * (64 - m) + b * m) + 32) >> 6)
static void blend_c(uint8_t *dst, const ptrdiff_t dst_stride, const uint8_t *tmp,
                    const int w, int h, const uint8_t *mask) {
    ~~for (; h-- > 0; dst += dst_stride, tmp += w, mask += w)~~
        for (int x = 0; x < w; x++) dst[x] = blend_px(dst[x], tmp[x], mask[x]);
}
```

```
  vle8.v v0, (a0)            // v0 = *dst;
  vle8.v v4, (a2)            // v4 = *tmp;
  vle8.v v8, (a5)            // v8 = *mask;
  vwmulu.vv v16, v4, v8      // v16 = v4*v8;
```

Example: 8bpc mc blend (inner loop)

```
#define blend_px(a, b, m) (((a * (64 - m) + b * m) + 32) >> 6)
static void blend_c(uint8_t *dst, const ptrdiff_t dst_stride, const uint8_t *tmp,
                    const int w, int h, const uint8_t *mask) {
    ~~for (; h-- > 0; dst += dst_stride, tmp += w, mask += w)~~
        for (int x = 0; x < w; x++) dst[x] = blend_px(dst[x], tmp[x], mask[x]);
}
```

```
vle8.v v0, (a0)          // v0 = *dst;
vle8.v v4, (a2)          // v4 = *tmp;
vle8.v v8, (a5)          // v8 = *mask;
vwmulu.vv v16, v4, v8    // v16 = v4*v8;
vrsub.vx v8, v8, t1      // v8 = 64 - v8;
```

RISE

Example: 8bpc mc blend (inner loop)

```
#define blend_px(a, b, m) (((a * (64 - m) + b * m) + 32) >> 6)
static void blend_c(uint8_t *dst, const ptrdiff_t dst_stride, const uint8_t *tmp,
                    const int w, int h, const uint8_t *mask) {
    ~~for (; h-- > 0; dst += dst_stride, tmp += w, mask += w)~~
        for (int x = 0; x < w; x++) dst[x] = blend_px(dst[x], tmp[x], mask[x]);
}
```

```
  vle8.v v0, (a0)           // v0 = *dst;
  vle8.v v4, (a2)           // v4 = *tmp;
  vle8.v v8, (a5)           // v8 = *mask;
  vwmulu.vv v16, v4, v8     // v16 = v4*v8;
  vrsub.vx v8, v8, t1       // v8 = 64 - v8;
  vwmaccu.vv v16, v0, v8    // v16 = v16 + v0*v8;
```

Example: 8bpc mc blend (inner loop)

```
#define blend_px(a, b, m) (((a * (64 - m) + b * m) + 32) >> 6)
static void blend_c(uint8_t *dst, const ptrdiff_t dst_stride, const uint8_t *tmp,
                    const int w, int h, const uint8_t *mask) {
    ~~for (; h-- > 0; dst += dst_stride, tmp += w, mask += w)~~
        for (int x = 0; x < w; x++) dst[x] = blend_px(dst[x], tmp[x], mask[x]);
}
```

```
  vle8.v v0, (a0)          // v0 = *dst;
  vle8.v v4, (a2)          // v4 = *tmp;
  vle8.v v8, (a5)          // v8 = *mask;
  vwmulu.vv v16, v4, v8    // v16 = v4*v8;
  vrsub.vx v8, v8, t1      // v8 = 64 - v8;
  vwmaccu.vv v16, v0, v8   // v16 = v16 + v0*v8;
  vnsra.wi v0, v16, 6      // v0 = (v16 + 32) >> 6;
```

RISE

Example: 8bpc mc blend (inner loop)

```
#define blend_px(a, b, m) (((a * (64 - m) + b * m) + 32) >> 6)
static void blend_c(uint8_t *dst, const ptrdiff_t dst_stride, const uint8_t *tmp,
                    const int w, int h, const uint8_t *mask) {
    ~~for (; h-- > 0; dst += dst_stride, tmp += w, mask += w)~~
        for (int x = 0; x < w; x++) dst[x] = blend_px(dst[x], tmp[x], mask[x]);
}
```

```
vle8.v v0, (a0)          // v0 = *dst;
vle8.v v4, (a2)          // v4 = *tmp;
vle8.v v8, (a5)          // v8 = *mask;
vwmulu.vv v16, v4, v8    // v16 = v4*v8;
vrsub.vx v8, v8, t1      // v8 = 64 - v8;
vwmaccu.vv v16, v0, v8   // v16 = v16 + v0*v8;
vnclipu.wi v0, v16, 6    // v0 = MAX(0, MIN(65536, (v16 + 32) >> 6));
vse8.v v0, (a0)          // *dst = v0;
```

Example: 8bpc mc blend (all together)

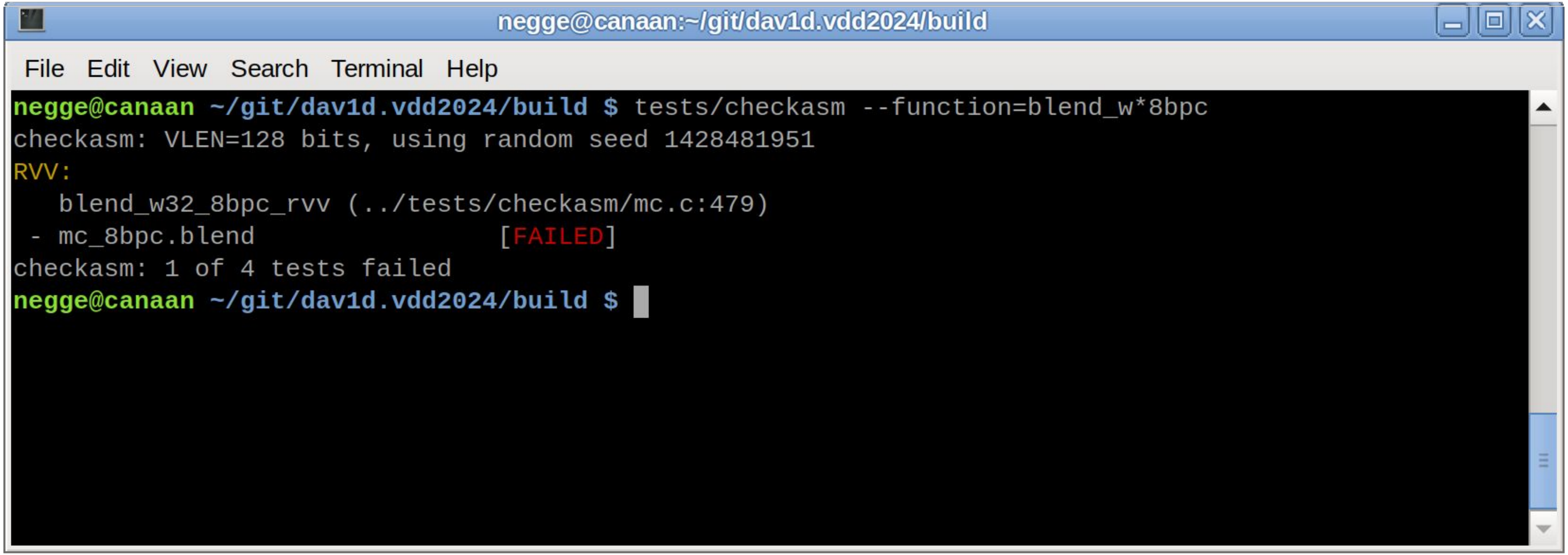
```c
#define blend_px(a, b, m) (((a * (64 - m) + b * m) + 32) >> 6)
static void blend_c(uint8_t *dst, const ptrdiff_t dst_stride, const uint8_t *tmp,
                    const int w, int h, const uint8_t *mask) {
    for (; h-- > 0; dst += dst_stride, tmp += w, mask += w)
        for (int x = 0; x < w; x++) dst[x] = blend_px(dst[x], tmp[x], mask[x]);
}
```

```
function blend_8bpc_rvv, export=1, ext="v”
  vsetvli zero, a3, e8, m1, ta, ma
  csrw vxrm, zero
  li t1, 64                  // t1 = 64;
1:                           // do {
  addi a4, a4, -1            //   h = h - 1;

  vle8.v v0, (a0)            // v0 = *dst;
  vle8.v v4, (a2)            // v4 = *tmp;
  vle8.v v8, (a5)            // v8 = *mask;

  vwmulu.vv v16, v4, v8      // v16 = v4*v8;
  vrsub.vx v8, v8, t1        // v8 = 64 - v8;
  vwmaccu.vv v16, v0, v8     // v16 = v16 + v0*v8;
  // v0 = MAX(0, MIN(65536, (v16 + 32) >> 6));
  vnclipu.wi v0, v16, 6
  vse8.v v0, (a0)            // *dst = v0;

  add a0, a0, a1             // dst += dst_stride
  add a2, a2, a3             // tmp += w;
  add a5, a5, a3             // mask += w;

  bnez a4, 1b                // } while (h != 0)
  ret
endfunc
```

RISE

Example: 8bpc mc blend (checkasm)

- Run `checkasm` to verify correctness
  - Passes at width of 4, 8 and 16 but fails when w = 32
  - What is going on?



negge@canaan:~/git/dav1d.vdd2024/build
File Edit View Search Terminal Help

```
negge@canaan ~/git/dav1d.vdd2024/build $ tests/checkasm --function=blend_w*8bpc
checkasm: VLEN=128 bits, using random seed 1428481951
RVV:
   blend_w32_8bpc_rvv (../tests/checkasm/mc.c:479)
 - mc_8bpc.blend                    [FAILED]
checkasm: 1 of 4 tests failed
negge@canaan ~/git/dav1d.vdd2024/build $
```

Example: 8bpc mc blend (outer loop)

```
#define blend_px(a, b, m) (((a * (64 - m) + b * m) + 32) >> 6)
static void blend_c(uint8_t *dst, const ptrdiff_t dst_stride, const uint8_t *tmp,
                    const int w, int h, const uint8_t *mask) {
    for (; h-- > 0; dst += dst_stride, tmp += w, mask += w)
        ~~for (int x = 0; x < w; x++) dst[x] = blend_px(dst[x], tmp[x], mask[x]);~~
}
```

```
function blend_8bpc_rvv, export=1, ext="v”
  vsetvli zero, a3, e8, m1, ta, ma
  li t1, 64                  // t1 = 64;
1:                           // do {
  addi a4, a4, -1            //   h = h - 1;
  ...




  bnez a4, 1b                // } while (h != 0)
  ret
endfunc
```

Canaan K230 has VLEN = 128
LMUL = m1 is not large enough for
8bpc * 32 = 256 bits

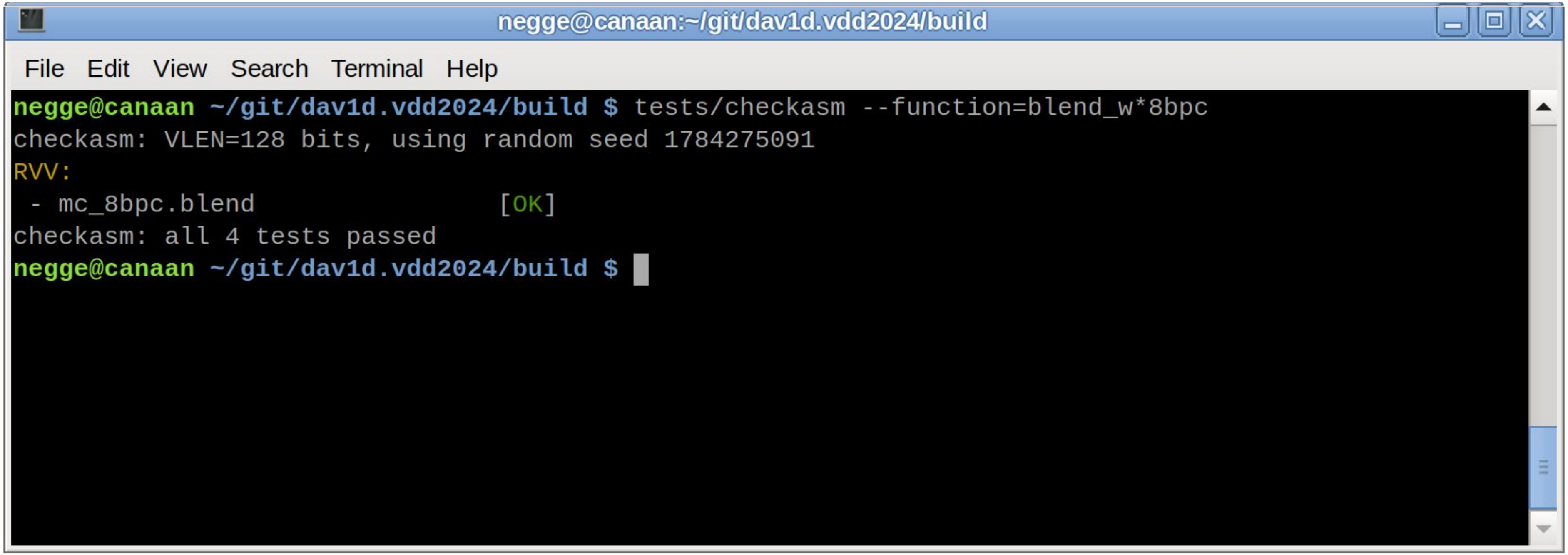
Example: 8bpc mc blend (outer loop)

```
#define blend_px(a, b, m) (((a * (64 - m) + b * m) + 32) >> 6)
static void blend_c(uint8_t *dst, const ptrdiff_t dst_stride, const uint8_t *tmp,
                    const int w, int h, const uint8_t *mask) {
    for (; h-- > 0; dst += dst_stride, tmp += w, mask += w)
        ~~for (int x = 0; x < w; x++) dst[x] = blend_px(dst[x], tmp[x], mask[x]);~~
}

function blend_8bpc_rvv, export=1, ext="v"
  vsetvli zero, a3, e8, m2, ta, ma
  li t1, 64                   // t1 = 64;
1:                            // do {
  addi a4, a4, -1             //   h = h - 1;
  ...




  bnez a4, 1b                 // } while (h != 0)
  ret
endfunc
```

Canaan K230 has VLEN = 128
LMUL = m1 is not large enough for
8bpc * 32 = 256 bits

Example: 8bpc mc blend (checkasm)

- Run `checkasm` to verify correctness
  - Passes at all widths, of 4, 8, 16 and 32

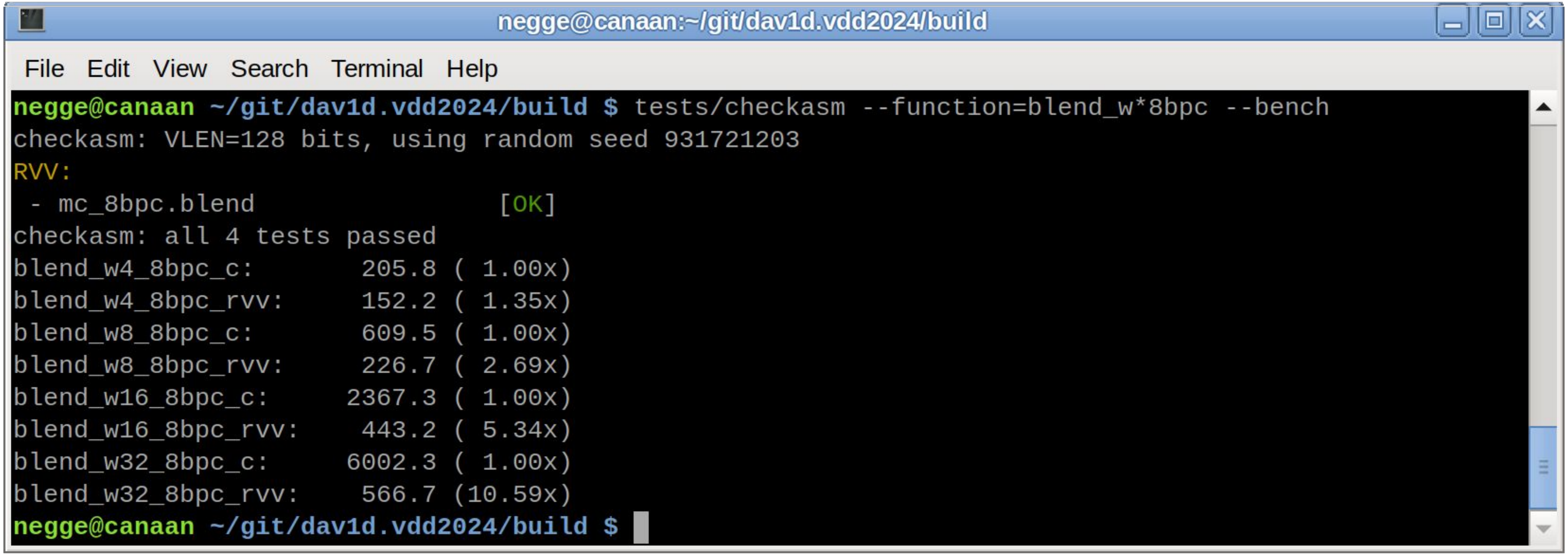


negge@canaan:~/git/dav1d.vdd2024/build
File Edit View Search Terminal Help

```
negge@canaan ~/git/dav1d.vdd2024/build $ tests/checkasm --function=blend_w*8bpc
checkasm: VLEN=128 bits, using random seed 1784275091
RVV:
 - mc_8bpc.blend                [OK]
checkasm: all 4 tests passed
negge@canaan ~/git/dav1d.vdd2024/build $ 
```

Example: 8bpc mc blend (checkasm)

- Run `checkasm` to verify performance
  - Is this right?

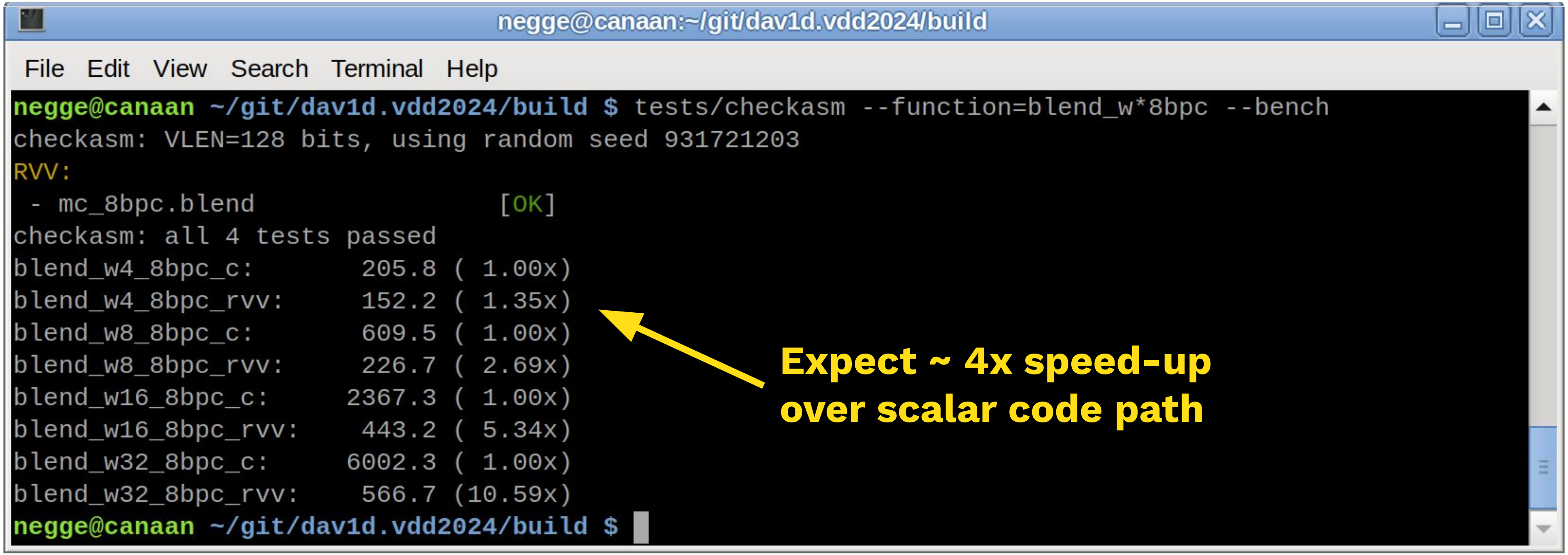


negge@canaan:~/git/dav1d.vdd2024/build
File Edit View Search Terminal Help

```
negge@canaan ~/git/dav1d.vdd2024/build $ tests/checkasm --function=blend_w*8bpc --bench
checkasm: VLEN=128 bits, using random seed 931721203
RVV:
 - mc_8bpc.blend                [OK]
checkasm: all 4 tests passed
blend_w4_8bpc_c:        205.8 ( 1.00x)
blend_w4_8bpc_rvv:      152.2 ( 1.35x)
blend_w8_8bpc_c:        609.5 ( 1.00x)
blend_w8_8bpc_rvv:      226.7 ( 2.69x)
blend_w16_8bpc_c:      2367.3 ( 1.00x)
blend_w16_8bpc_rvv:     443.2 ( 5.34x)
blend_w32_8bpc_c:      6002.3 ( 1.00x)
blend_w32_8bpc_rvv:     566.7 (10.59x)
negge@canaan ~/git/dav1d.vdd2024/build $
```

Example: 8bpc mc blend (checkasm)

- Run `checkasm` to verify performance
  - Is this right?
  - Good performance at w = 32, but w = 4 is definitely not right


negge@canaan:~/git/dav1d.vdd2024/build
File Edit View Search Terminal Help

```
negge@canaan ~/git/dav1d.vdd2024/build $ tests/checkasm --function=blend_w*8bpc --bench
checkasm: VLEN=128 bits, using random seed 931721203
RVV:
 - mc_8bpc.blend                [OK]
checkasm: all 4 tests passed
blend_w4_8bpc_c:        205.8 ( 1.00x)
blend_w4_8bpc_rvv:      152.2 ( 1.35x)
blend_w8_8bpc_c:        609.5 ( 1.00x)
blend_w8_8bpc_rvv:      226.7 ( 2.69x)
blend_w16_8bpc_c:      2367.3 ( 1.00x)
blend_w16_8bpc_rvv:     443.2 ( 5.34x)
blend_w32_8bpc_c:      6002.3 ( 1.00x)
blend_w32_8bpc_rvv:     566.7 (10.59x)
negge@canaan ~/git/dav1d.vdd2024/build $
```

Expect ~ 4x speed-up over scalar code path

Example: 8bpc mc blend (preamble)

```
#define blend_px(a, b, m) (((a * (64 - m) + b * m) + 32) >> 6)
static void blend_c(uint8_t *dst, const ptrdiff_t dst_stride, const uint8_t *tmp,
                    const int w, int h, const uint8_t *mask) {
    for (; h-- > 0; dst += dst_stride, tmp += w, mask += w)
        ~~for (int x = 0; x < w; x++) dst[x] = blend_px(dst[x], tmp[x], mask[x]);~~
}

function blend_8bpc_rvv, export=1, ext="v”
  vsetvli zero, a3, e8, m2, ta, ma

  ret
endfunc
```

Example: 8bpc mc blend (preamble)

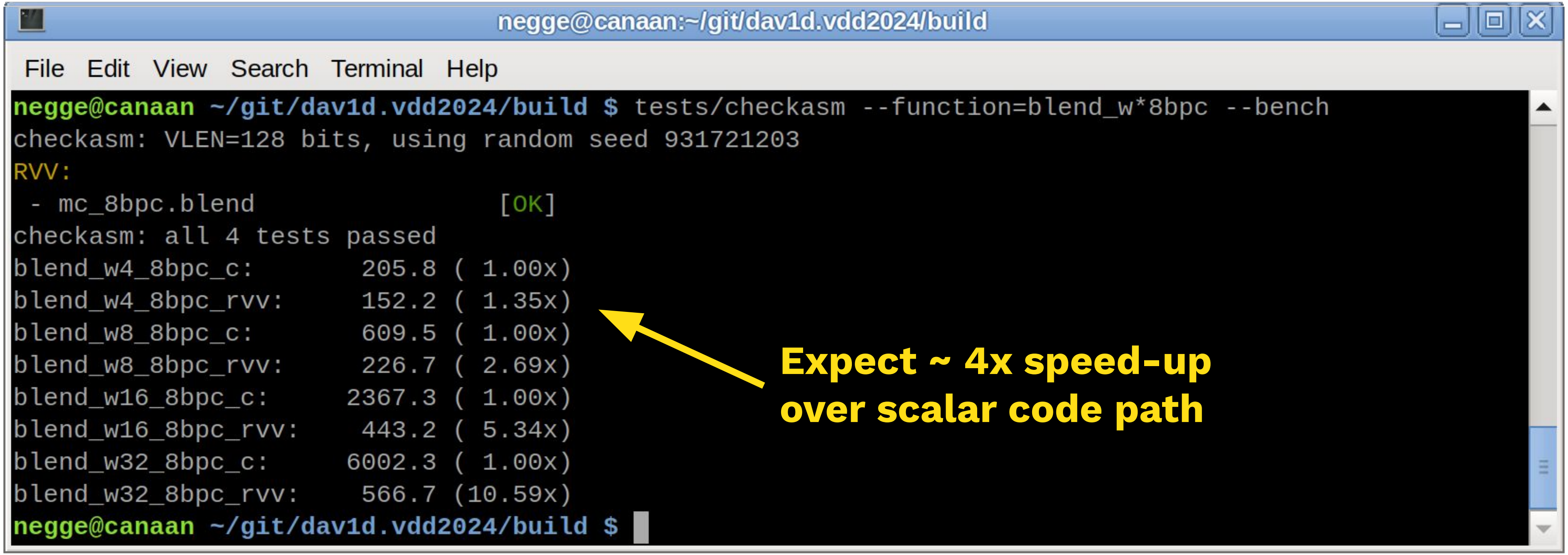
```
#define blend_px(a, b, m) (((a * (64 - m) + b * m) + 32) >> 6)
static void blend_c(uint8_t *dst, const ptrdiff_t dst_stride, const uint8_t *tmp,
                    const int w, int h, const uint8_t *mask) {
    for (; h-- > 0; dst += dst_stride, tmp += w, mask += w)
        ~~for (int x = 0; x < w; x++) dst[x] = blend_px(dst[x], tmp[x], mask[x]);~~
}
```

```
function blend_8bpc_rvv, export=1, ext="v”
    li t0, 4
    beq a3, t0, 4f
    li t0, 8
    beq a3, t0, 8f
    li t0, 16
    beq a3, t0, 16f
32: vsetvli zero, a3, e8, m2, ta, ma
     j L(blend_epilog)
16: vsetvli zero, a3, e8, m1, ta, ma
    j L(blend_epilog)
8:  vsetvli zero, a3, e8, mf2, ta, ma
    j L(blend_epilog)
4:  vsetvli zero, a3, e8, mf4, ta, ma
```

```
L(blend_epilog):
  csrw vxrm, zero
  li t1, 64                 // t1 = 64;

1:                          // do {
  addi a4, a4, -1           //   h = h - 1;
  ...
  add a0, a0, a1            // dst += dst_stride
  add a2, a2, a3            // tmp += w;
  add a5, a5, a3            // mask += w;

  bnez a4, 1b               // } while (h != 0)
  ret
endfunc
```

Example: 8bpc mc blend (checkasm)

- Run `checkasm` to verify performance
  - Is this right?
  - Good performance at w = 32, but w = 4 is definitely not right



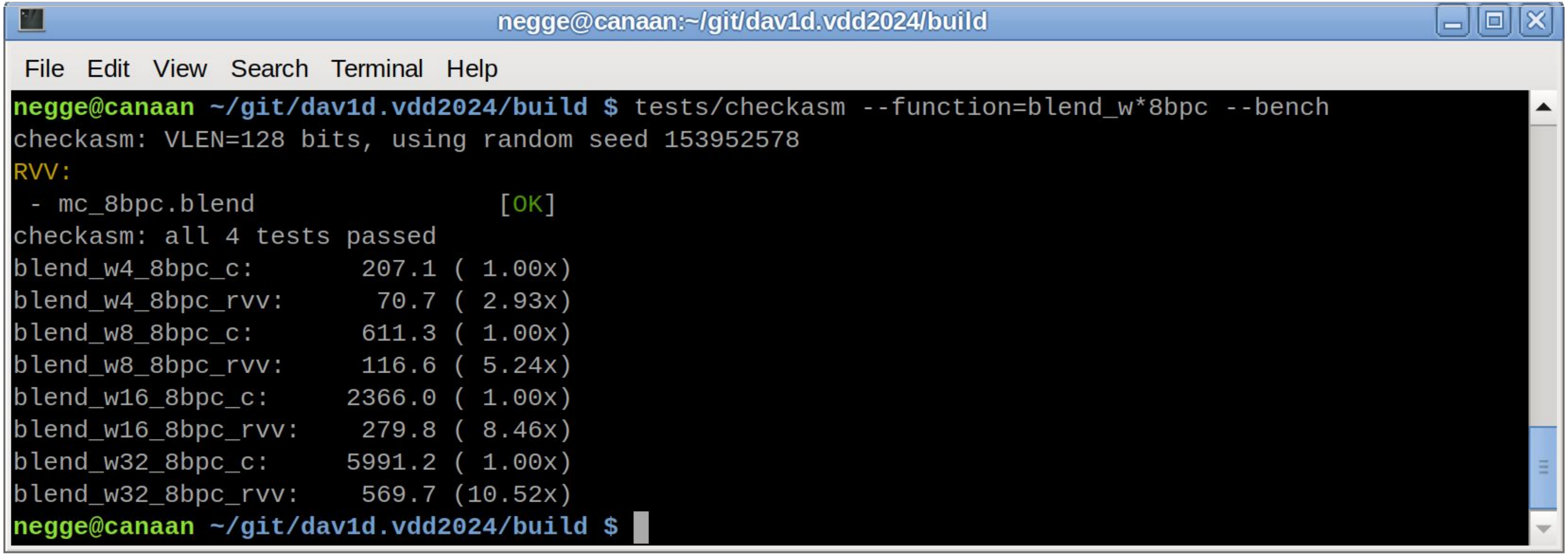
negge@canaan:~/git/dav1d.vdd2024/build
File Edit View Search Terminal Help

```
negge@canaan ~/git/dav1d.vdd2024/build $ tests/checkasm --function=blend_w*8bpc --bench
checkasm: VLEN=128 bits, using random seed 931721203
RVV:
 - mc_8bpc.blend                [OK]
checkasm: all 4 tests passed
blend_w4_8bpc_c:         205.8 ( 1.00x)
blend_w4_8bpc_rvv:       152.2 ( 1.35x)
blend_w8_8bpc_c:         609.5 ( 1.00x)
blend_w8_8bpc_rvv:       226.7 ( 2.69x)
blend_w16_8bpc_c:       2367.3 ( 1.00x)
blend_w16_8bpc_rvv:      443.2 ( 5.34x)
blend_w32_8bpc_c:       6002.3 ( 1.00x)
blend_w32_8bpc_rvv:      566.7 (10.59x)
negge@canaan ~/git/dav1d.vdd2024/build $
```

Expect ~ 4x speed-up over scalar code path

Example: 8bpc mc blend (checkasm)

- Run `checkasm` to verify performance
  - Looking much better now...
  - But what happens when run on larger VLEN?

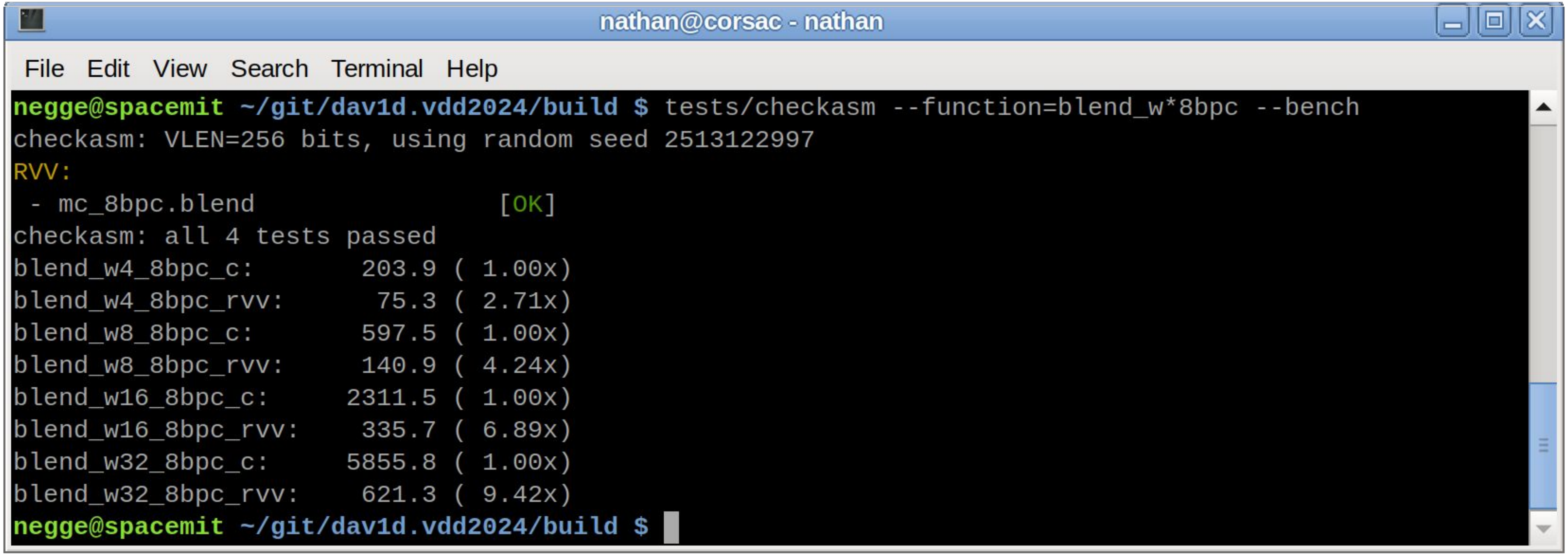


negge@canaan:~/git/dav1d.vdd2024/build
File Edit View Search Terminal Help

```
negge@canaan ~/git/dav1d.vdd2024/build $ tests/checkasm --function=blend_w*8bpc --bench
checkasm: VLEN=128 bits, using random seed 153952578
RVV:
 - mc_8bpc.blend                [OK]
checkasm: all 4 tests passed
blend_w4_8bpc_c:         207.1 ( 1.00x)
blend_w4_8bpc_rvv:        70.7 ( 2.93x)
blend_w8_8bpc_c:         611.3 ( 1.00x)
blend_w8_8bpc_rvv:       116.6 ( 5.24x)
blend_w16_8bpc_c:       2366.0 ( 1.00x)
blend_w16_8bpc_rvv:      279.8 ( 8.46x)
blend_w32_8bpc_c:       5991.2 ( 1.00x)
blend_w32_8bpc_rvv:      569.7 (10.52x)
negge@canaan ~/git/dav1d.vdd2024/build $
```

Example: 8bpc mc blend (checkasm)

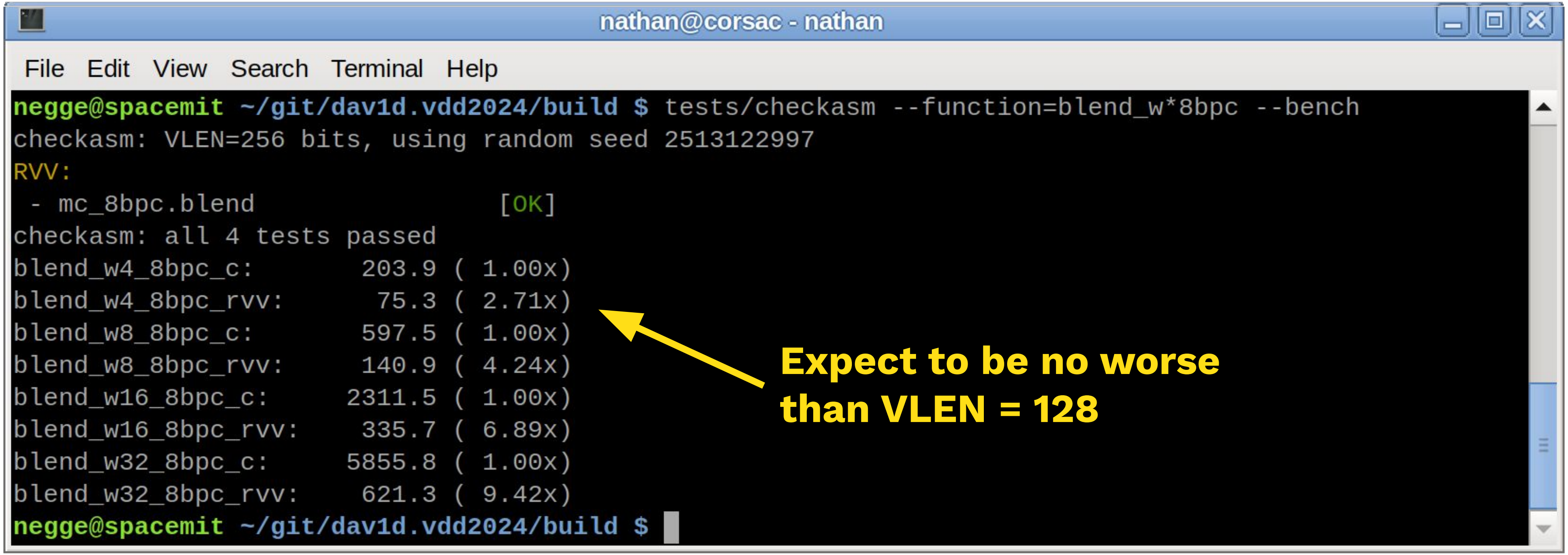
- Run `checkasm` on VLEN=256 to verify performance
  - Is this right?



```
negge@spacemit ~/git/dav1d.vdd2024/build $ tests/checkasm --function=blend_w*8bpc --bench
checkasm: VLEN=256 bits, using random seed 2513122997
RVV:
 - mc_8bpc.blend                [OK]
checkasm: all 4 tests passed
blend_w4_8bpc_c:        203.9 ( 1.00x)
blend_w4_8bpc_rvv:       75.3 ( 2.71x)
blend_w8_8bpc_c:        597.5 ( 1.00x)
blend_w8_8bpc_rvv:      140.9 ( 4.24x)
blend_w16_8bpc_c:      2311.5 ( 1.00x)
blend_w16_8bpc_rvv:     335.7 ( 6.89x)
blend_w32_8bpc_c:      5855.8 ( 1.00x)
blend_w32_8bpc_rvv:     621.3 ( 9.42x)
negge@spacemit ~/git/dav1d.vdd2024/build $
```

Example: 8bpc mc blend (checkasm)

- Run `checkasm` on VLEN=256 to verify performance
  - Is this right?
  - Worse performance with larger SIMD, doesn't seem right



```
negge@spacemit ~/git/dav1d.vdd2024/build $ tests/checkasm --function=blend_w*8bpc --bench
checkasm: VLEN=256 bits, using random seed 2513122997
RVV:
 - mc_8bpc.blend                [OK]
checkasm: all 4 tests passed
blend_w4_8bpc_c:        203.9 ( 1.00x)
blend_w4_8bpc_rvv:       75.3 ( 2.71x)
blend_w8_8bpc_c:        597.5 ( 1.00x)
blend_w8_8bpc_rvv:      140.9 ( 4.24x)
blend_w16_8bpc_c:      2311.5 ( 1.00x)
blend_w16_8bpc_rvv:     335.7 ( 6.89x)
blend_w32_8bpc_c:      5855.8 ( 1.00x)
blend_w32_8bpc_rvv:     621.3 ( 9.42x)
negge@spacemit ~/git/dav1d.vdd2024/build $
```

Expect to be no worse than VLEN = 128

Example: 8bpc mc blend (VLEN=256 preamble)

```
#define blend_px(a, b, m) (((a * (64 - m) + b * m) + 32) >> 6)
static void blend_c(uint8_t *dst, const ptrdiff_t dst_stride, const uint8_t *tmp,
                    const int w, int h, const uint8_t *mask) {
    for (; h-- > 0; dst += dst_stride, tmp += w, mask += w)
        ~~for (int x = 0; x < w; x++) dst[x] = blend_px(dst[x], tmp[x], mask[x]);~~
```

```
function blend_8bpc_rvv, ext="v”
    li t0, 4
    beq a3, t0, 4f
    li t0, 8
    beq a3, t0, 8f
    li t0, 16
    beq a3, t0, 16f
32: vsetvli zero, a3, e8, m2, ta, ma
     j L(blend_epilog)
16: vsetvli zero, a3, e8, m1, ta, ma
    j L(blend_epilog)
8:  vsetvli zero, a3, e8, mf2, ta, ma
    j L(blend_epilog)
4:  vsetvli zero, a3, e8, mf4, ta, ma
L(blend_epilog):
    ...
```

Example: 8bpc mc blend (VLEN=256 preamble)

```
#define blend_px(a, b, m) (((a * (64 - m) + b * m) + 32) >> 6)
static void blend_c(uint8_t *dst, const ptrdiff_t dst_stride, const uint8_t *tmp,
                    const int w, int h, const uint8_t *mask) {
    for (; h-- > 0; dst += dst_stride, tmp += w, mask += w)
        ~~for (int x = 0; x < w; x++) dst[x] = blend_px(dst[x], tmp[x], mask[x]);~~
```

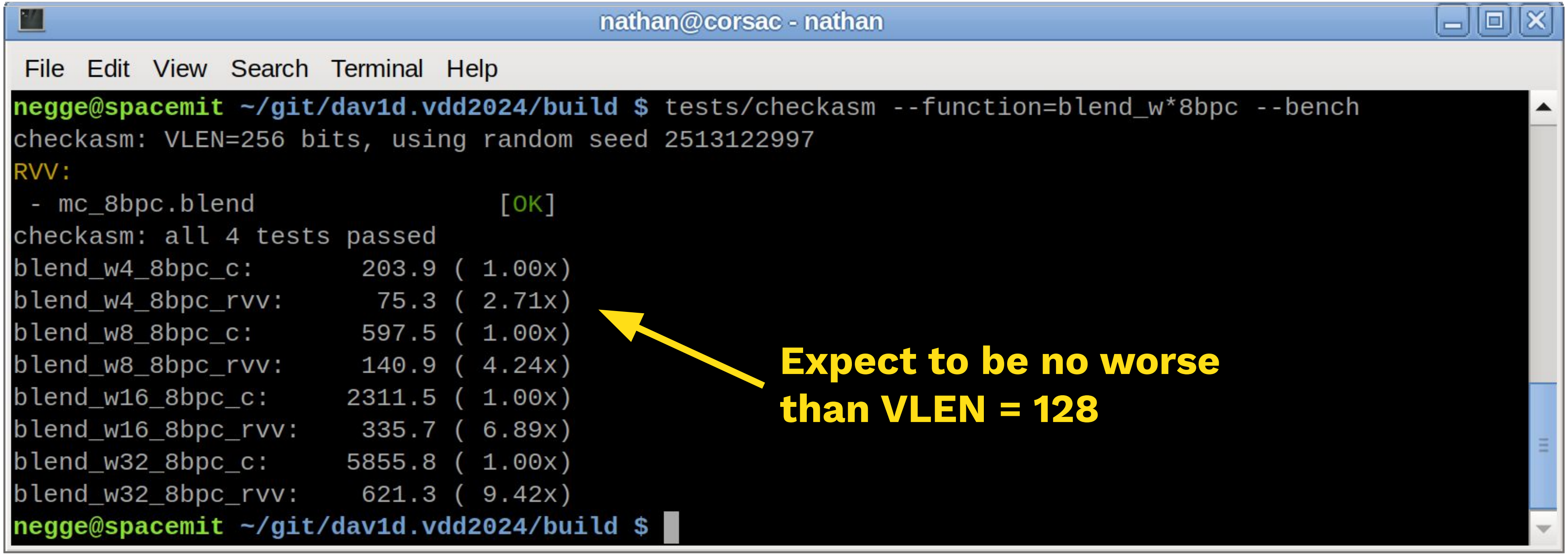
```
function blend_8bpc_rvv, ext="v”
    li t0, 4
    beq a3, t0, 4f
    li t0, 8
    beq a3, t0, 8f
    li t0, 16
    beq a3, t0, 16f
32: vsetvli zero, a3, e8, m2, ta, ma
     j L(blend_epilog)
16: vsetvli zero, a3, e8, m1, ta, ma
    j L(blend_epilog)
8:  vsetvli zero, a3, e8, mf2, ta, ma
    j L(blend_epilog)
4:  vsetvli zero, a3, e8, mf4, ta, ma
L(blend_epilog):
    ...
```

```
function blend_vl256_8bpc_rvv, ext="v”
    li t0, 4
    beq a3, t0, 4f
    li t0, 8
    beq a3, t0, 8f
    li t0, 16
    beq a3, t0, 16f
32: vsetvli zero, a3, e8, m1, ta, ma
     j L(blend_epilog)
16: vsetvli zero, a3, e8, mf2, ta, ma
    j L(blend_epilog)
8:  vsetvli zero, a3, e8, mf4, ta, ma
    j L(blend_epilog)
4:  vsetvli zero, a3, e8, mf8, ta, ma
    j L(blend_epilog)
endfunc
```

RISE

Example: 8bpc mc blend (checkasm)

- Run `checkasm` on VLEN=256 to verify performance
  - Is this right?
  - Worse performance with larger SIMD, doesn’t seem right

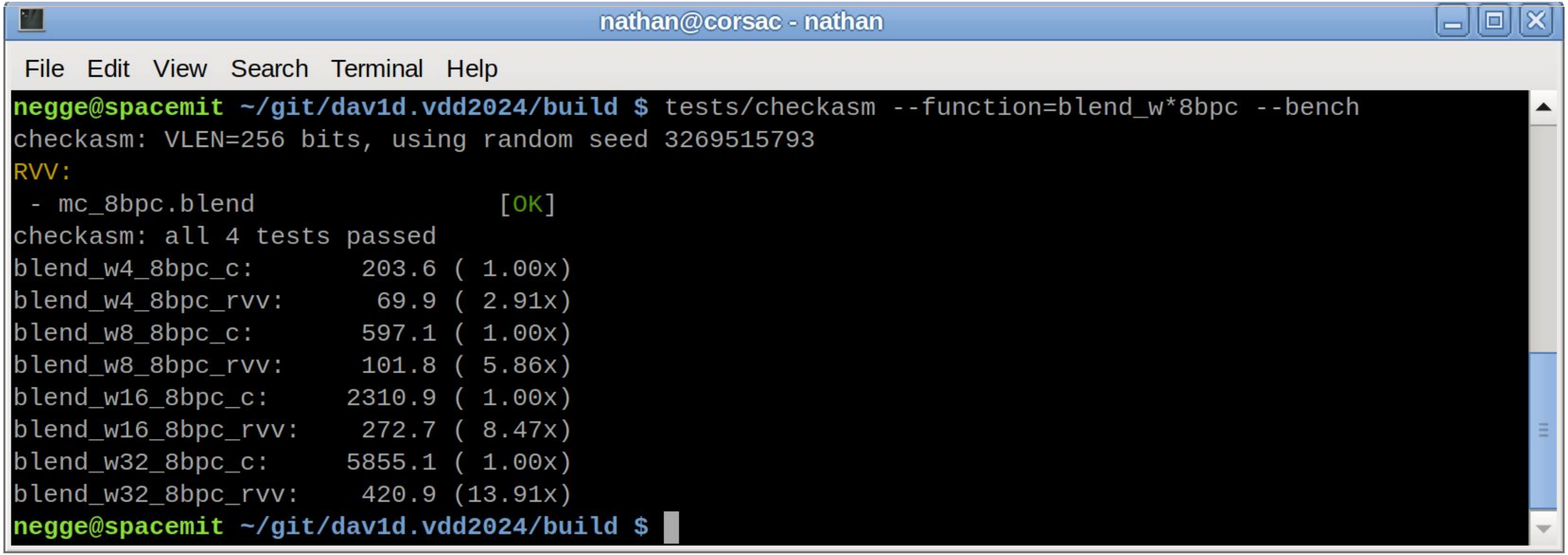


```
negge@spacemit ~/git/dav1d.vdd2024/build $ tests/checkasm --function=blend_w*8bpc --bench
checkasm: VLEN=256 bits, using random seed 2513122997
RVV:
 - mc_8bpc.blend                [OK]
checkasm: all 4 tests passed
blend_w4_8bpc_c:        203.9 ( 1.00x)
blend_w4_8bpc_rvv:       75.3 ( 2.71x)
blend_w8_8bpc_c:        597.5 ( 1.00x)
blend_w8_8bpc_rvv:      140.9 ( 4.24x)
blend_w16_8bpc_c:      2311.5 ( 1.00x)
blend_w16_8bpc_rvv:     335.7 ( 6.89x)
blend_w32_8bpc_c:      5855.8 ( 1.00x)
blend_w32_8bpc_rvv:     621.3 ( 9.42x)
negge@spacemit ~/git/dav1d.vdd2024/build $
```

Expect to be no worse than VLEN = 128

Example: 8bpc mc blend (checkasm)

- Run `checkasm` on VLEN=256 to verify performance
  - Looks much better…
  - Is this as good as we get?



```
negge@spacemit ~/git/dav1d.vdd2024/build $ tests/checkasm --function=blend_w*8bpc --bench
checkasm: VLEN=256 bits, using random seed 3269515793
RVV:
 - mc_8bpc.blend                [OK]
checkasm: all 4 tests passed
blend_w4_8bpc_c:        203.6 ( 1.00x)
blend_w4_8bpc_rvv:       69.9 ( 2.91x)
blend_w8_8bpc_c:        597.1 ( 1.00x)
blend_w8_8bpc_rvv:      101.8 ( 5.86x)
blend_w16_8bpc_c:      2310.9 ( 1.00x)
blend_w16_8bpc_rvv:     272.7 ( 8.47x)
blend_w32_8bpc_c:      5855.1 ( 1.00x)
blend_w32_8bpc_rvv:     420.9 (13.91x)
negge@spacemit ~/git/dav1d.vdd2024/build $ 
```

Example: 8bpc mc blend (unroll inner loop)

```
#define blend_px(a, b, m) (((a * (64 - m) + b * m) + 32) >> 6)
static void blend_c(uint8_t *dst, const ptrdiff_t dst_stride, const uint8_t *tmp,
                    const int w, int h, const uint8_t *mask) {
    ~~for (; h-- > 0; dst += dst_stride, tmp += w, mask += w)~~
        for (int x = 0; x < w; x++) dst[x] = blend_px(dst[x], tmp[x], mask[x]);
}
```

```
  vle8.v v4, (a2)           // v4 = *tmp;
  add a2, a2, a3            // tmp += w;
  vle8.v v6, (a2)           // v4 = *tmp;
  add a2, a2, a3            // tmp += w;

  vle8.v v8, (a5)           // v8 = *mask;
  add a5, a5, a3            // mask += w;
  vle8.v v10, (a5)          // v10 = *mask;
  add a5, a5, a3            // mask += w;

  vle8.v v0, (a0)           // v0 = *dst;
  // t0 = dst + dst_stride
  add t0, a0, a1
  vle8.v v2, (t0)           // v2 = *t0;
```

RISE

Example: 8bpc mc blend (unroll inner loop)

```
#define blend_px(a, b, m) (((a * (64 - m) + b * m) + 32) >> 6)
static void blend_c(uint8_t *dst, const ptrdiff_t dst_stride, const uint8_t *tmp,
                    const int w, int h, const uint8_t *mask) {
    ~~for (; h-- > 0; dst += dst_stride, tmp += w, mask += w)~~
        for (int x = 0; x < w; x++) dst[x] = blend_px(dst[x], tmp[x], mask[x]);
}
```

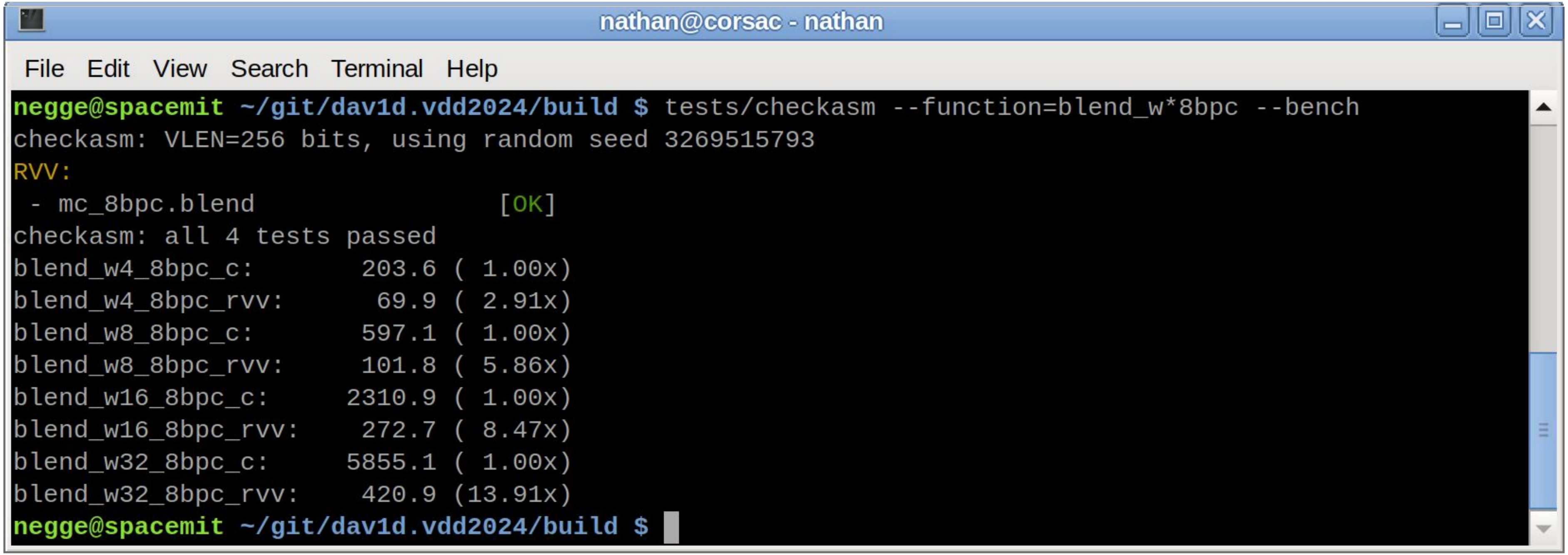
```
vle8.v v4, (a2)          // v4 = *tmp;
add a2, a2, a3           // tmp += w;
vle8.v v6, (a2)          // v4 = *tmp;
add a2, a2, a3           // tmp += w;

vle8.v v8, (a5)          // v8 = *mask;
add a5, a5, a3           // mask += w;
vle8.v v10, (a5)         // v10 = *mask;
add a5, a5, a3           // mask += w;

vle8.v v0, (a0)          // v0 = *dst;
// t0 = dst + dst_stride
add t0, a0, a1
vle8.v v2, (t0)          // v2 = *t0;
```

```
vwmulu.vv v16, v4, v8    // v16 = v4*v8;
vwmulu.vv v20, v6, v10   // v20 = v6*v10;
vrsub.vx v8, v8, t1      // v8 = 64 - v8;
vrsub.vx v10, v10, t1    // v10 = 64 - v10;
vwmaccu.vv v16, v0, v8   // v16 = v16 + v0*v8
vwmaccu.vv v20, v2, v10  // v20 = v20 + v2*v10
// v0 = MAX(0, MIN(65536, (v16 + 32) >> 6));
vnclipu.wi v0, v16, 6
// v2 = MAX(0, MIN(65536, (v20 + 32) >> 6));
vnclipu.wi v2, v20, 6
vse8.v v0, (a0)          // *dst = v0;
vse8.v v2, (t0)          // *t0 = v2;
// dst = t0 + dst_stride
add a0, t0, a1
```

RISE

Example: 8bpc mc blend (checkasm)

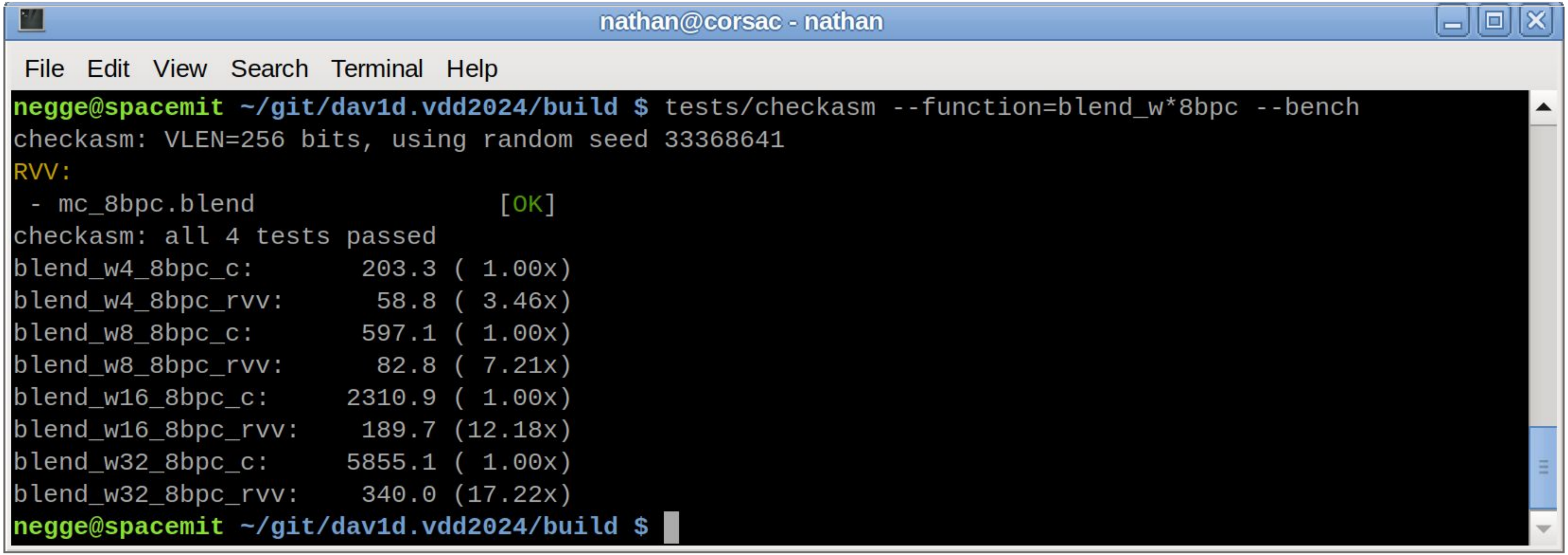
- Run checkasm on VLEN=256 to verify performance
  - Looks much better...
  - Is this as good as we get?



```
negge@spacemit ~/git/dav1d.vdd2024/build $ tests/checkasm --function=blend_w*8bpc --bench
checkasm: VLEN=256 bits, using random seed 3269515793
RVV:
 - mc_8bpc.blend                [OK]
checkasm: all 4 tests passed
blend_w4_8bpc_c:        203.6 ( 1.00x)
blend_w4_8bpc_rvv:       69.9 ( 2.91x)
blend_w8_8bpc_c:        597.1 ( 1.00x)
blend_w8_8bpc_rvv:      101.8 ( 5.86x)
blend_w16_8bpc_c:      2310.9 ( 1.00x)
blend_w16_8bpc_rvv:     272.7 ( 8.47x)
blend_w32_8bpc_c:      5855.1 ( 1.00x)
blend_w32_8bpc_rvv:     420.9 (13.91x)
negge@spacemit ~/git/dav1d.vdd2024/build $
```

Example: 8bpc mc blend (checkasm)

- Run `checkasm` on VLEN=256 with unrolled loops to verify performance
  - Looking quite good



```
negge@spacemit ~/git/dav1d.vdd2024/build $ tests/checkasm --function=blend_w*8bpc --bench
checkasm: VLEN=256 bits, using random seed 33368641
RVV:
 - mc_8bpc.blend                [OK]
checkasm: all 4 tests passed
blend_w4_8bpc_c:         203.3 ( 1.00x)
blend_w4_8bpc_rvv:        58.8 ( 3.46x)
blend_w8_8bpc_c:         597.1 ( 1.00x)
blend_w8_8bpc_rvv:        82.8 ( 7.21x)
blend_w16_8bpc_c:       2310.9 ( 1.00x)
blend_w16_8bpc_rvv:      189.7 (12.18x)
blend_w32_8bpc_c:       5855.1 ( 1.00x)
blend_w32_8bpc_rvv:      340.0 (17.22x)
negge@spacemit ~/git/dav1d.vdd2024/build $
```

Example: 8bpc mc blend (VLEN=256 preamble)

```
#define blend_px(a, b, m) (((a * (64 - m) + b * m) + 32) >> 6)
static void blend_c(uint8_t *dst, const ptrdiff_t dst_stride, const uint8_t *tmp,
                    const int w, int h, const uint8_t *mask) {
    for (; h-- > 0; dst += dst_stride, tmp += w, mask += w)
        ~~for (int x = 0; x < w; x++) dst[x] = blend_px(dst[x], tmp[x], mask[x]);~~
```

```
function blend_8bpc_rvv, ext="v"
    li t0, 4
    beq a3, t0, 4f
    li t0, 8
    beq a3, t0, 8f
    li t0, 16
    beq a3, t0, 16f
32: vsetvli zero, a3, e8, m2, ta, ma
     j L(blend_epilog)
16: vsetvli zero, a3, e8, m1, ta, ma
    j L(blend_epilog)
8:  vsetvli zero, a3, e8, mf2, ta, ma
    j L(blend_epilog)
4:  vsetvli zero, a3, e8, mf4, ta, ma
L(blend_epilog):
    ...
```

```
function blend_vl256_8bpc_rvv, ext="v"
    li t0, 4
    beq a3, t0, 4f
    li t0, 8
    beq a3, t0, 8f
    li t0, 16
    beq a3, t0, 16f
32: vsetvli zero, a3, e8, m1, ta, ma
     j L(blend_epilog)
16: vsetvli zero, a3, e8, mf2, ta, ma
    j L(blend_epilog)
8:  vsetvli zero, a3, e8, mf4, ta, ma
    j L(blend_epilog)
4:  vsetvli zero, a3, e8, mf8, ta, ma
    j L(blend_epilog)
endfunc
```

Example: 8bpc mc blend (VLEN=256 preamble)

- Assume Zbb extension present when RVV 1.0 detected
- Rewrite into branchless code using ctz
  - This works because vsew[5:3] = 0 when SEW = e8

```
function blend_8bpc_rvv, ext="v”
    ctz t0, a3
    addi t0, t0, 0xc4
L(blend_epilog):
    andi t0, t0, 0xc7
    vsetvl zero, a3, t0
    ...
    ret
endfunc
```

```
function blend_vl256_8bpc_rvv, ext="v”
    ctz t0, a3
    addi t0, t0, 0xc3
    j L(blend_epilog)
endfunc
```

only 10 bytes for VLEN = 256 VLS routine!

Experiment: RISC-V Vector v Arm NEON

- Run RVV implemented 2D transforms on hardware
  - Kendryte K230
    - Single Core @ 1.6 GHz
    - RVV 1.0 with VLEN = **128 bit**
    - 32k L1 / 256kB L2 / 512MB DDR3
  - ODROID C2
    - Quad Core A53 @ 1.5 GHz
    - Advanced SIMD, aka NEON with **128 bit** registers
    - 32kB L1 / 512kB L2 / 2GB DDR3
- Collect C and ASM timings, compare deltas**

** Warning, not a perfect comparison

- ARM uses `pmccntr_el0` for timings, RISC-V uses `clock_gettime()`
- Differences in CPU frequencies, L2 cache, memory
- Close enough for scalar -> vector verification

Experiment: RISC-V Vector v Arm NEON

```
NEON:
 - mc_8bpc.blend                            [OK]
checkasm: all 4 tests passed
blend_w4_8bpc_c:          332.5 ( 1.00x)
blend_w4_8bpc_neon:        66.8 ( 4.98x)
blend_w8_8bpc_c:         1043.3 ( 1.00x)
blend_w8_8bpc_neon:       114.2 ( 9.14x)
blend_w16_8bpc_c:        3855.7 ( 1.00x)
blend_w16_8bpc_neon:      299.2 (12.89x)
blend_w32_8bpc_c:        9563.5 ( 1.00x)
blend_w32_8bpc_neon:      725.8 (13.18x)

NEON:
 - mc_16bpc.blend                           [OK]
checkasm: all 4 tests passed
blend_w4_16bpc_c:         334.8 ( 1.00x)
blend_w4_16bpc_neon:       73.0 ( 4.59x)
blend_w8_16bpc_c:                1044.3 (
1.00x)
blend_w8_16bpc_neon:      134.3 ( 7.78x)
blend_w16_16bpc_c:       3860.9 ( 1.00x)
blend_w16_16bpc_neon:     478.3 ( 8.07x)
blend_w32_16bpc_c:       9576.2 ( 1.00x)
blend_w32_16bpc_neon:    1227.7 ( 7.80x)
```

RISE

Conclusions

- Benchmarks are a *powerful* tool and **essential** when developing performance optimizations

- Test on multiple VLEN to ensure performance working as expected

- Understand the impact of LMUL on throughput (<-- this is critical!) and specialize on VLEN where possible

- Balance between LMUL register pressure and loop unrolling
  - Often worth unrolling once to improve throughput

- Always test performance on *representative* hardware
  - Access to more RVV implementations needed for verification!

- Possible to do ISA <-> ISA and VLEN <-> VLEN comparisons

- Use the latest compilers, toolchains, binutils, etc. when testing

Questions?